Lecture 5: Data Model 설계 3

김 강 희
khkim@ssu.ac.kr

목 차

- 이론:
  - abstract 타입
  - interface 타입
- 실습:
  - v1: Tetris 클래스 외부에 interface 타입으로 8개 핸들러 정의
  - v2: Tetris 클래스 외부에 interface 타입으로 key별 동작 정의
  - v3: 과제

이번 Lecture에서 알아야 할 사항들

- 확장가능한 Tetris 클래스 설계 (부제: 조립식 코딩)
  - 클래스 내부의 Data를 외부에서 변경함 → Tetris::init()
  - 클래스 내부의 Logic을 외부에서 변경함 → 이번 강의 내용
  - 클래스 정의의 구체화 → 클래스 확장 범위를 한계짓기 위함

    : "Tetris 게임은 한정된 2차원 영역 안에서 임의 모양의 블록을 대상으로 사용자 key 마다 정해진 동작(do)을 수행하고, 충돌 발생시 복구 동작(undo)을 수행하며 오래 버티면 이기는 게임"
- 문법
  - Interface 타입이 언제 필요한가?
  - Interface 타입을 어떤 클래스 내부와 외부에서 정의할 때 차이점은 무엇인가?

Abstract Class 타입

- 자바 언어는 한 클래스의 선언 안에서 하나 이상의 메쏘드들의 정의를 포함하지 않는 것을 허용하고 있음
- 이것은 해당 클래스와 정의 없는 메쏘드들을 abstract로 선언함으로써 가능함
- abstract 클래스는 인스턴스를 생성할 수 없으며, 그 서브클래스는 반드시 abstract 메쏘드들을 모두 정의하여 클래스를 완성하거나 (그래야 인스턴스 생성 가능), 아니면 서브클래스 역시 abstract 클래스로 선언되어야 함
- 미래의 개발자가 abstract class를 완성해서 사용할 것을 기대하면서 abstract class를 정의해야 함

```
public abstract class Incomplete {
    public void run() { preprocessing(); postprocessing(); }
    private void preprocessing() {    …  // do something   }
    public abstract void postprocessing();
}
public class Complete extends Incomplete {
    public void postprocessing() {    … // do something else    }
}
```

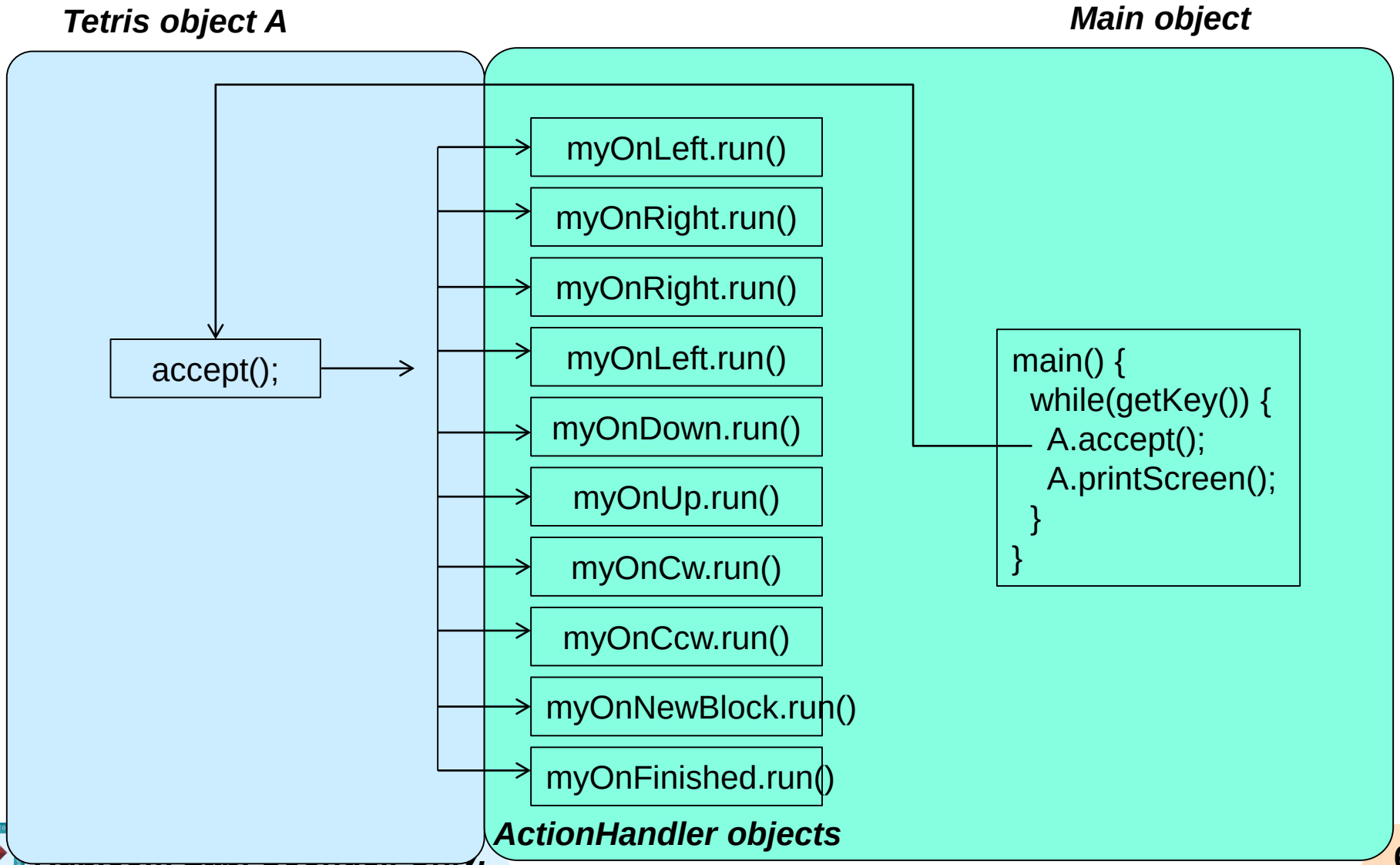
Interface 타입

- 자바 언어는 단순함을 추구하여 다중 상속을 지원하지 않음
  - 다중 상속이란 한 클래스가 여러 개의 부모 클래스로부터 상속받는 것을 말함
- 대신에, 자바 언어는 interface 개념을 사용하여 간접적으로 다중 상속의 개념을 지원하고 있음

```
public interface DoHandler     { void do(char key); }
public interface UndoHandler   { void undo(char key); }
public class Action implements DoHandler, UndoHandler {
      public void do(char key)        {    …    }
      public void undo(char key)      {    …    }
}
```

- interface란 abstract 메쏘드들만을 가진 abstract 클래스로 이해할 수 있음
- interface를 이용하면, generic code + plugin code 구조를 구현할 수 있음
  - 하나의 generic code만을 두기 때문에 memory footprint가 작아지고 디버깅이 용이해짐
  - Plugin code를 통해서 어떤 클래스의 내부 코드를 외부에서 변경하는 것이 가능해짐

Main.java & Tetris.java (v1)



Main.java (v1)

```
184     public static void main2(String[] args) throws Exception {
185         char key;
186         TetrisState state;
187         Tetris.init(setOfBlockArrays);
188
189         OnLeft myOnLeft = new OnLeft() {
190             public void run(Tetris t, char key) throws Exception { t.left = t.left - 2; }
191         };
192         OnRight myOnRight = new OnRight() {
193             public void run(Tetris t, char key) throws Exception { t.left = t.left + 2; }
194         };
195         OnDown myOnDown = new OnDown() {
196             public void run(Tetris t, char key) throws Exception { t.top = t.top + 1; }
197         };
198         OnUp myOnUp = new OnUp() {
199             public void run(Tetris t, char key) throws Exception { t.top = t.top - 1; }
200         };
201         OnCw myOnCw = new OnCw() {
202             public void run(Tetris t, char key) throws Exception {
203                 t.idxBlockDegree = (t.idxBlockDegree+1)%t.nBlockDegrees;
204                 t.currBlk = t.setOfBlockObjects[t.idxBlockType][t.idxBlockDegree];
205             }
206         };
207         OnCcw myOnCcw = new OnCcw() {
208             public void run(Tetris t, char key) throws Exception {
209                 t.idxBlockDegree = (t.idxBlockDegree+t.nBlockDegrees-1)%t.nBlockDegrees;
210                 t.currBlk = t.setOfBlockObjects[t.idxBlockType][t.idxBlockDegree];
211             }
212         };
```

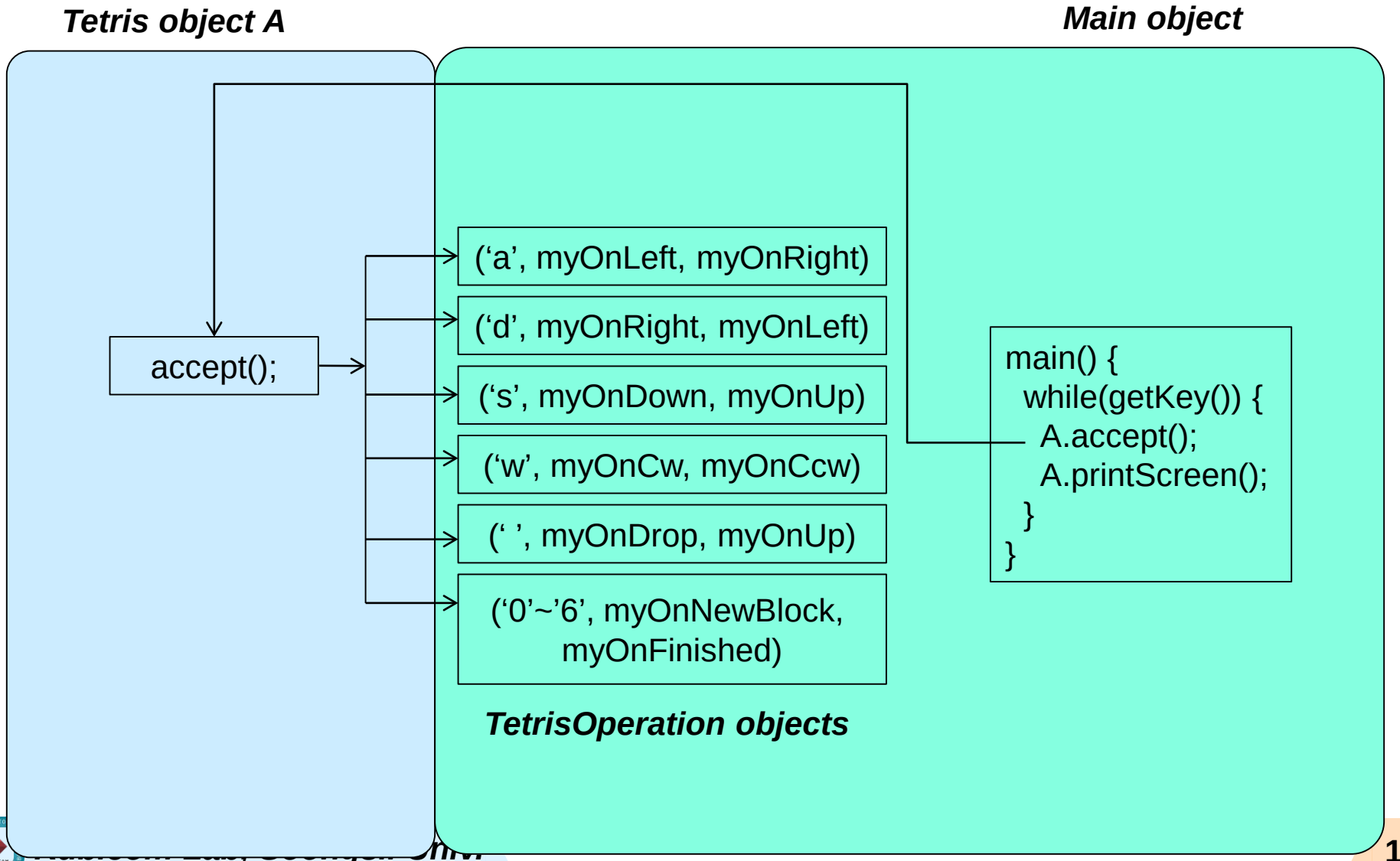
Main.java (v1)

```
213         OnNewBlock myOnNewBlock = new OnNewBlock() {
214             public void run(Tetris t, char key) throws Exception {
215                 t.oScreen = deleteFullLines(t.oScreen, t.currBlk, t.top, t.iScreenDy, t.iScreenDx, t.iScreenDw);
216                 t.iScreen.paste(t.oScreen, top: 0, left: 0);
217                 t.idxBlockType = key - '0';
218                 t.idxBlockDegree = 0;
219                 t.currBlk = t.setOfBlockObjects[t.idxBlockType][t.idxBlockDegree];
220                 t.top = 0;
221                 t.left = t.iScreenDw + t.iScreenDx/2 - (t.currBlk.get_dx()+1)/2;
222             }
223             private Matrix deleteFullLines(Matrix screen, Matrix blk, int top, int dy, int dx, int dw) throws Exception {
224                 Matrix line, zero, temp;
225                 if (blk == null) return screen;
226                 int cy, y, nDeleted = 0,nScanned = blk.get_dy();
227                 if (top + blk.get_dy() - 1 >= dy)
228                     nScanned -= (top + blk.get_dy() - dy);
229                 zero = new Matrix( cy: 1, cx: dx - 2*dw);
230                 for (y = nScanned - 1; y >= 0 ; y--) {
231                     cy = top + y + nDeleted;
232                     line = screen.clip(cy, left: 0, bottom: cy + 1, screen.get_dx());
233                     if (line.sum() == screen.get_dx()) {
234                         temp = screen.clip( top: 0, left: 0, cy, screen.get_dx());
235                         screen.paste(temp, top: 1, left: 0);
236                         screen.paste(zero, top: 0, dw);
237                         nDeleted++;
238                     }
239                 }
240                 return screen;
241             }
242         };
```

Main.java (v1)

```
243         OnFinished myOnFinished = new OnFinished() {
244             public void run(Tetris t, char key) throws Exception {
245                 System.out.println("OnFinished.run() called");
246             }
247         };
248         Tetris.setOnLeftListener(myOnLeft);
249         Tetris.setOnRightListener(myOnRight);
250         Tetris.setOnDownListener(myOnDown);
251         Tetris.setOnUpListener(myOnUp);
252         Tetris.setOnCwListener(myOnCw);
253         Tetris.setOnCcwListener(myOnCcw);
254         Tetris.setOnNewBlockListener(myOnNewBlock);
255         Tetris.setOnFinishedListener(myOnFinished);
256         Tetris board = new Tetris( cy: 15,  cx: 10);
257         Random random = new Random();
258         key = (char) ('0' + random.nextInt( bound: 7));
259         board.accept(key);
260         board.printScreen(); System.out.println();
261
262         while ((key = getKey()) != 'q') {
263             state = board.accept(key);
264             board.printScreen(); System.out.println();
265             if (state == TetrisState.NewBlock) {
266                 key = (char) ('0' + random.nextInt( bound: 7));
267                 state = board.accept(key);
268                 board.printScreen(); System.out.println();
269                 if (state == TetrisState.Finished) break; // Game Over!
270             }
271         }
272         System.out.println("Program terminated!");
273     }
```

Main.java & Tetris.java (v2)



Main.java (v2)

```
187     public static void main(String[] args) throws Exception {
188         char key;
189         TetrisState state;
190         Tetris.init(setOfBlockArrays);
191
192         OnLeft myOnLeft = new OnLeft() {
193             public boolean run(Tetris t, char key) throws Exception {
194                 t.left = t.left - 1;
195                 return t.anyConflict( updateNeeded: true);
196             }
197         };
198         OnRight myOnRight = new OnRight() {
199             public boolean run(Tetris t, char key) throws Exception {
200                 t.left = t.left + 1;
201                 return t.anyConflict( updateNeeded: true);
202             }
203         };
204         OnDown myOnDown = new OnDown() {
205             public boolean run(Tetris t, char key) throws Exception {
206                 t.top = t.top + 1;
207                 return t.anyConflict( updateNeeded: true);
208             }
209         };
210         OnUp myOnUp = new OnUp() {
211             public boolean run(Tetris t, char key) throws Exception {
212                 t.top = t.top - 1;
213                 return t.anyConflict( updateNeeded: true);
214             }
215         };
```

Main.java (v2)

```
216     OnDrop myOnDrop = new OnDrop() {
217         public boolean run(Tetris t, char key) throws Exception {
218             do { t.top = t.top + 1; }
219             while (t.anyConflict( updateNeeded: false) == false);
220             return t.anyConflict( updateNeeded: true);
221         }
222     };
223     OnCw myOnCw = new OnCw() {
224         public boolean run(Tetris t, char key) throws Exception {
225             t.idxBlockDegree = (t.idxBlockDegree + 1) % t.nBlockDegrees;
226             t.currBlk = t.setOfBlockObjects[t.idxBlockType][t.idxBlockDegree];
227             return t.anyConflict( updateNeeded: true);
228         }
229     };
230
231     OnCcw myOnCcw = new OnCcw() {
232         public boolean run(Tetris t, char key) throws Exception {
233             t.idxBlockDegree = (t.idxBlockDegree + t.nBlockDegrees - 1) % t.nBlockDegrees;
234             t.currBlk = t.setOfBlockObjects[t.idxBlockType][t.idxBlockDegree];
235             return t.anyConflict( updateNeeded: true);
236         }
237     };
```

Main.java (v2)

```
238  OnNewBlock myOnNewBlock = new OnNewBlock() {
239      public boolean run(Tetris t, char key) throws Exception {
240          t.oScreen = deleteFullLines(t.oScreen, t.currBlk, t.top, t.iScreenDy, t.iScreenDx, t.iScreenDw);
241          t.iScreen = new Matrix(t.oScreen);
242          t.idxBlockType = key - '0';
243          t.idxBlockDegree = 0;
244          t.currBlk = t.setOfBlockObjects[t.idxBlockType][t.idxBlockDegree];
245          t.top = 0;
246          t.left = t.iScreenDw + t.iScreenDx / 2 - (t.currBlk.get_dx()+1) / 2;
247          return t.anyConflict( updateNeeded: true);
248      }
249      private Matrix deleteFullLines(Matrix screen, Matrix blk, int top, int dy, int dx, int dw) throws Exception {
250          Matrix line, zero, temp;
251          if (blk == null) // called right after the game starts.
252              return screen; // no lines to be deleted
253          int cy, y, nDeleted = 0, nScanned = blk.get_dy();
254          if (top + blk.get_dy() - 1 >= dy)
255              nScanned -= (top + blk.get_dy() - dy);
256          zero = new Matrix( cy: 1,  cx: dx - 2*dw);
257          for (y = nScanned - 1; y >= 0; y--) {
258              cy = top + y + nDeleted;
259              line = screen.clip(cy,  left: 0,  bottom: cy + 1, screen.get_dx());
260              if (line.sum() == screen.get_dx()) {
261                  temp = screen.clip( top: 0,  left: 0, cy, screen.get_dx());
262                  screen.paste(temp,  top: 1,  left: 0);
263                  screen.paste(zero,  top: 0, dw);
264                  nDeleted++;
265              }
266          }
267          return screen;
268      }
269  };
```

Main.java (v2)

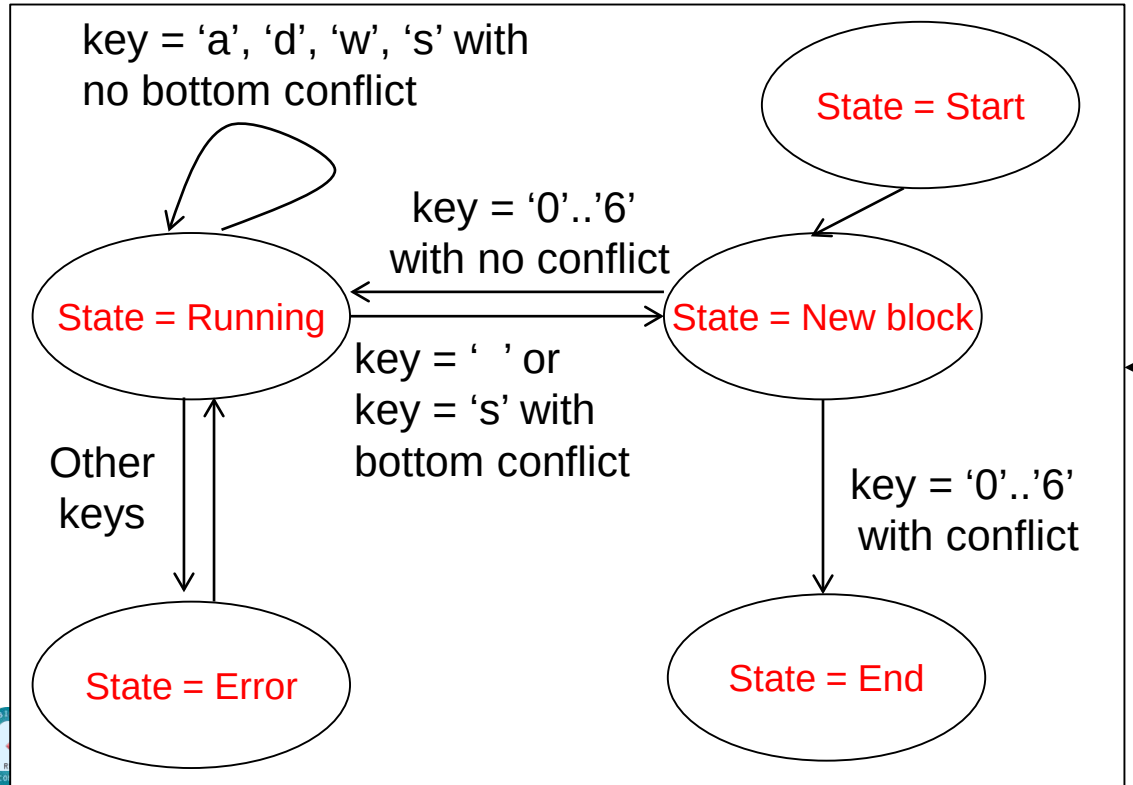
```
270
271 OnFinished myOnFinished = new OnFinished() {
272     public boolean run(Tetris t, char key) throws Exception {
273         System.out.println("OnFinished.run() called");
274         return false;
275     }
};
276 Tetris.setOperation( key: 'a', Running, myOnLeft, Running, myOnRight, Running);
277 Tetris.setOperation( key: 'd', Running, myOnRight, Running, myOnLeft, Running);
278 Tetris.setOperation( key: 's', Running, myOnDown, Running, myOnUp, NewBlock);
279 Tetris.setOperation( key: 'w', Running, myOnCw, Running, myOnCcw, Running);
280 Tetris.setOperation( key: ' ', Running, myOnDrop, Running, myOnUp, NewBlock);
281 Tetris.setOperation( key: '0', NewBlock, myOnNewBlock, Running, myOnFinished, Finished);
282 Tetris.setOperation( key: '1', NewBlock, myOnNewBlock, Running, myOnFinished, Finished);
283 Tetris.setOperation( key: '2', NewBlock, myOnNewBlock, Running, myOnFinished, Finished);
284 Tetris.setOperation( key: '3', NewBlock, myOnNewBlock, Running, myOnFinished, Finished);
285 Tetris.setOperation( key: '4', NewBlock, myOnNewBlock, Running, myOnFinished, Finished);
286 Tetris.setOperation( key: '5', NewBlock, myOnNewBlock, Running, myOnFinished, Finished);
287 Tetris.setOperation( key: '6', NewBlock, myOnNewBlock, Running, myOnFinished, Finished);
288
289 Tetris board = new Tetris( cy: 15,  cx: 10);
290 Random random = new Random();
291 key = (char) ('0' + random.nextInt( bound: 7));
292 board.accept(key);
293 board.printScreen(); System.out.println();
294
```

Main.java (v2)

```
295             while ((key = getKey()) != 'q') {
296                 state = board.accept(key);
297                 board.printScreen(); System.out.println();
298                 if (state == NewBlock) {
299                     key = (char) ('0' + random.nextInt( bound: 7));
300                     state = board.accept(key);
301                     board.printScreen(); System.out.println();
302                     if (state == Finished) break; // Game Over!
303                 }
304             }
305             System.out.println("Program terminated!");
306         }
```

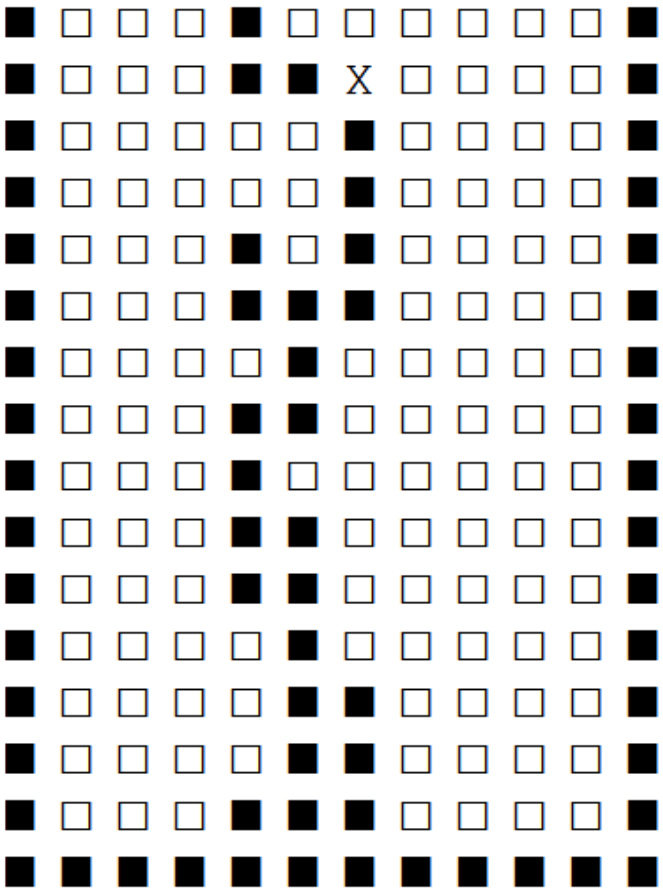
상태 기계

- 테트리스 게임의 상태 기계 모델
  - key 값과 idxType 값의 나열을 하나의 입력 시퀀스(input sequence)로 이해하고 Tetris 상태 기계는 Start, Running, New Block, End, Error 상태를 가진다고 이해할 수 있음
  - 동일한 입력 시퀀스에 대해서 상태 기계는 항상 동일한 상태를 가짐
  - Tetris 상태 기계의 입력을 하나의 변수 타입으로 통일하면 Tetris class 코드 상에 상태 기계 관점을 더 잘 표현할 수 있음



3, a, w, SPC, 0, d s, SPC, 5, …

실행 결과 (이전과 동일함)

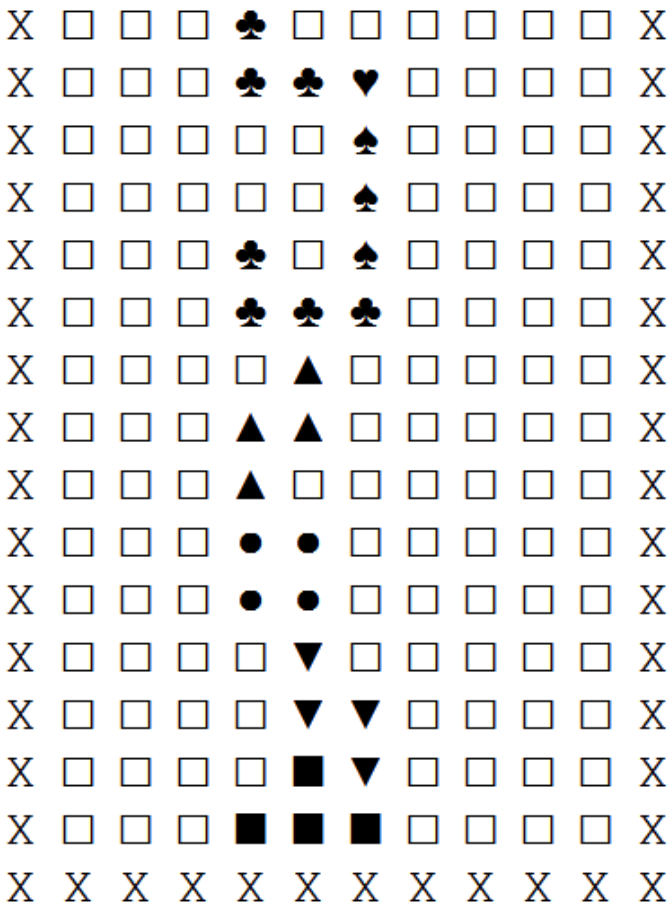


좋은 코드의 특징

- 가독성
  - 하나의 변수의 하나의 의미를, 하나의 함수는 하나의 기능을 표현
  - 변수와 함수 이름이 서술적일수록 좋음 (예: “변수 = 동사(변수)”)
- 모듈화
  - 가정(assumption)과 파생 로직의 분리(예: hardcoded constants)
  - 인터페이스와 내부 구현의 명확한 분리
- 간결성
  - 중복된 로직이 없고 필수적인 최소 로직만 존재
- 확장성
  - Generic code와 plugin code의 분리
- 이식성
  - 시스템 함수들과 사용자 함수들의 명확한 분리
- 성능
  - 성능 최적화가 필요한 common case가 코드 상에서 쉽게 식별될 수 있으면 충분함

남은 숙제

- Tetris class를 상속받아 CTetris class를 작성할 것
  - Tetris class의 private method들 중에서 필요한 것은 protected로 변경하고, public & protected method들 중에서 재정의 필요한 것들을 최소로 하여 CTetris class를 작성함 (accept method는 재정의하지 말 것!!)
  - OnCLeft, OnCRight, OnCDown, OnCUp, OnCCw, OnCCcw, OnCNewBlock, OnCFinalized 등 플러그인 코드 방식을 그대로 유지해야 함



감사합니다!